ce-10.
（Cプログラミング応用，全１４回）

https://www.kkaneko.jp/cc/c/index.html

金子邦彦

ce-10. 二分探索木

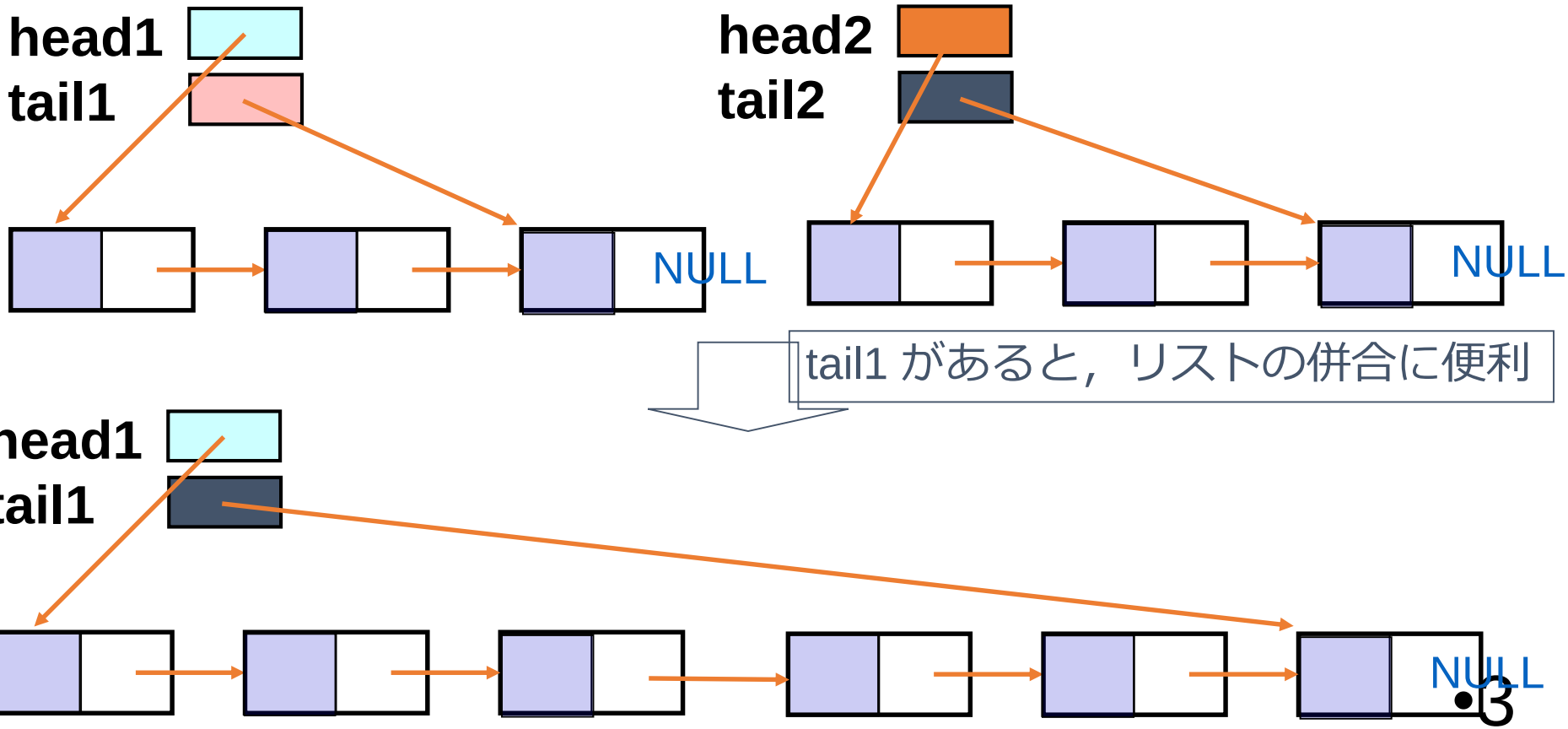
（C プログラミング応用）（全１４回）

URL: https://www.kkaneko.jp/pro/c/index.html

金子邦彦

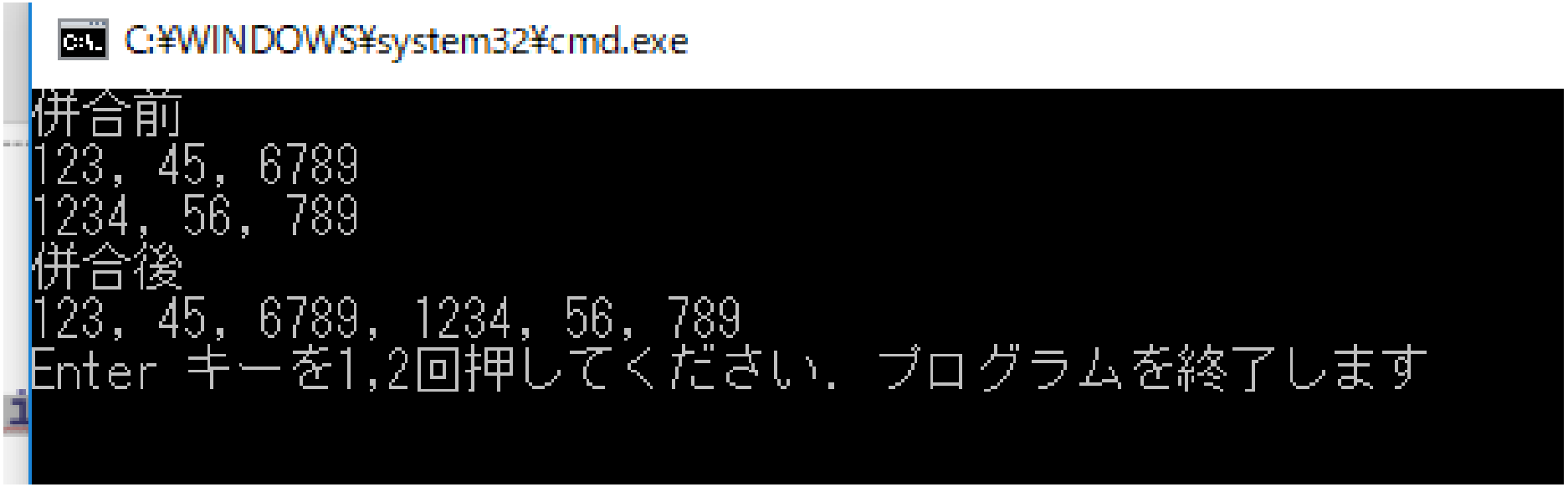
例題１．リストの併合

- ２つのリストを併合するプログラムを動かしてみる



```
#include "stdio.h"
#include <math.h>
struct data_list {
  int data;
  struct data_list *next;
};
void print_list(struct data_list *p)
{
  struct data_list *x;
  if ( p == NULL ) {
    return;
  }
  printf( "%d", p->data );
  x = p->next;
  while( x != NULL ) {
    printf( ", %d", x->data );
    x = x-> next;
  }
  printf( "¥n" );
}
struct data_list *new_list(int x)
{
  struct data_list *c = new(struct data_list);
  c->data = x;
  c->next = NULL;
  return c;
}
void insert_list(struct data_list *p, int x)
{
  struct data_list *c = new(struct data_list);
  c->data = p->data;
  c->next = p->next;
  p->data = x;
  p->next = c;
}
int _tmain()
{
  int ch;
  struct data_list *head1;
  struct data_list *tail1;
  struct data_list *head2;
  struct data_list *tail2;
  head1 = new_list( 6789 );
  insert_list( head1, 45 );
  tail1 = head1->next; // ２個目の要素を作った時点で head1->next と tail1 が等しい
  insert_list( head1, 123 );
  head2 = new_list( 789 );
  insert_list( head2, 56 );
  tail2 = head2->next; // ２個目の要素を作った時点で head2->next と tail2 が等しい
  insert_list( head2, 1234 );
  printf( "併合前¥n" );
  print_list( head1 );
  print_list( head2 );
  //
  tail1->next = head2;
  tail1 = tail2;
  printf( "併合後¥n" );
  print_list( head1 );
  printf( "Enter キーを1,2回押してください. プログラムを終了します¥n");
  ch = getchar();
  ch = getchar();
  return 0;
}
```

```
tail1->next = head2;
tail1 = tail2;
```


C:¥WINDOWS¥system32¥cmd.exe
併合前
123, 45, 6789
1234, 56, 789
併合後
123, 45, 6789, 1234, 56, 789
Enter キーを1,2回押してください．プログラムを終了します

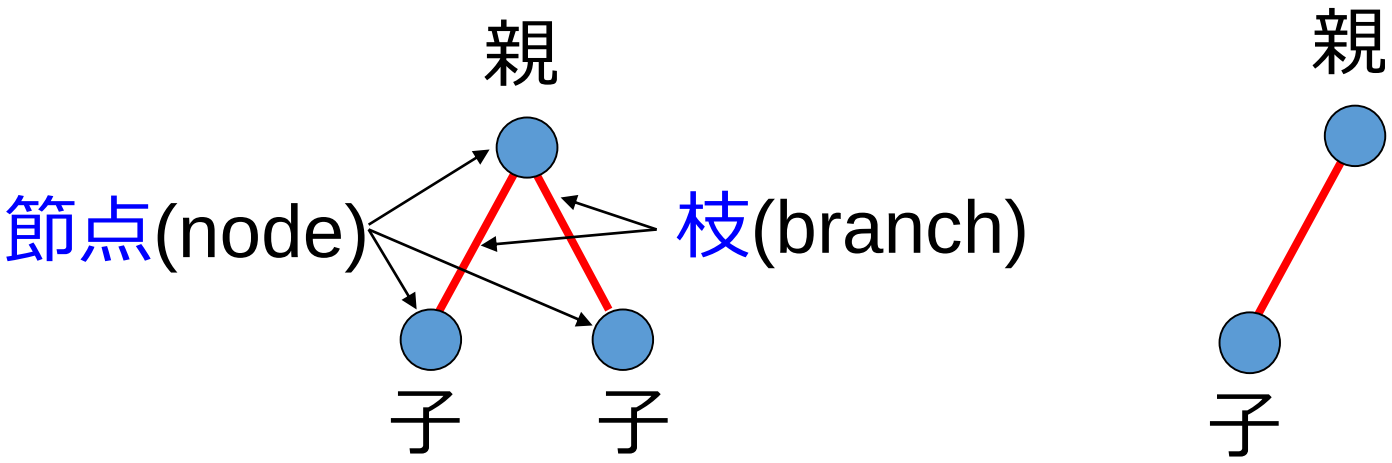
実行結果の例

二分探索木 (binary search tree)

- 整列可能な任意個数のデータ集合に対し、効率良い探索（二分探索）を行うためのデータ構造
- 整列前：35 46 21 13 61 40 69
- 整列後（昇順）：13 21 35 40 46 61 69
- 節点から出る枝が高々２本の木構造（二分木）を作り、各節点にデータを置く

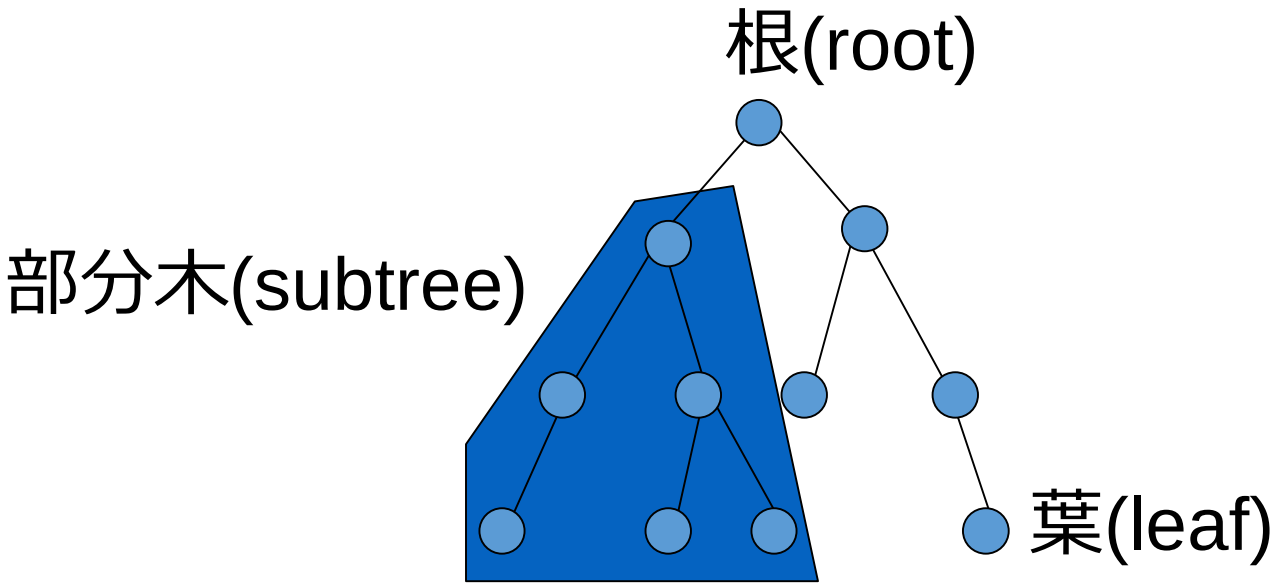
二分探索木の構造

- 節点(node)と枝(branch)
  - 枝はデータ間の（親子）関係を表す
  - 節点に入る（親からの）枝は１本
  - 節点から出ていく（子への）枝は０または１または２本



二分探索木の構造（つづき）

- 根(root)： 親を持たない節点
- 葉(leaf)： 子を持たない節点
- 部分木(subtree)： 任意の節点から下のすべての枝と節点

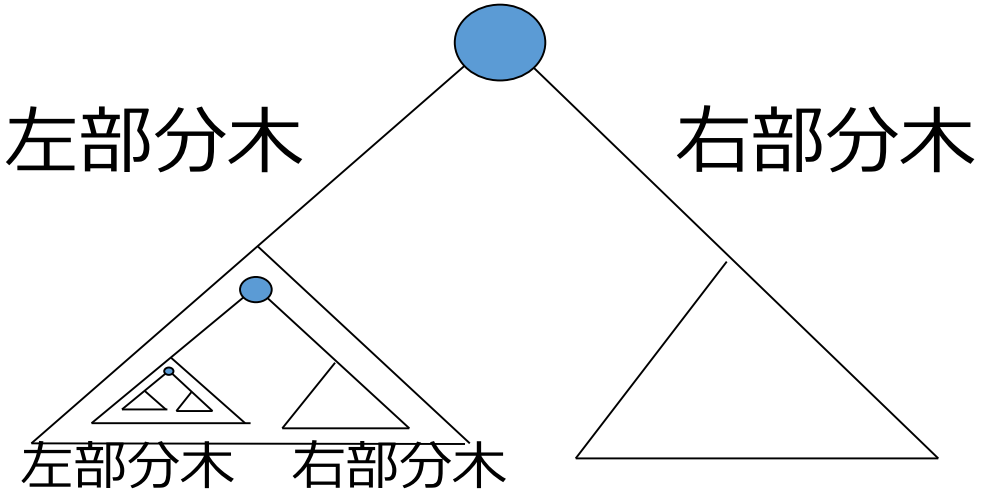


二分探索木の再帰的構造

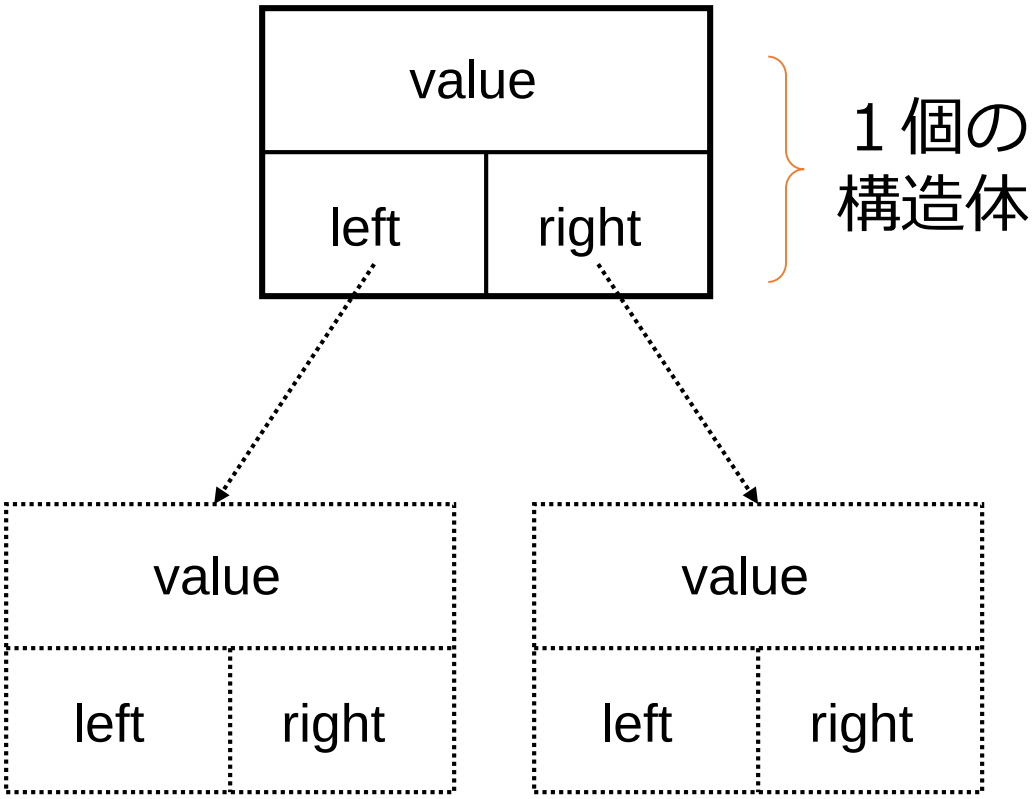
- 左部分木、右部分木も二分木をなす
- 親節点に対する処理が、子節点に対する処理に帰着されることが多い

（例）探索において，子へ移動し，親での処理と同様の処理を繰り返す

→再帰関数を用いて自然に表現できる。



二分探索木の各ノードを C言語の構造体で表現




```
// 2分木のノード
struct  BTNode
{
      BTNode *left;
      BTNode *right;
  int value;
};
```

int value の部分は，格納すべきデータに応じて変わる

二分探索木の節点(node) の生成関数 new_node

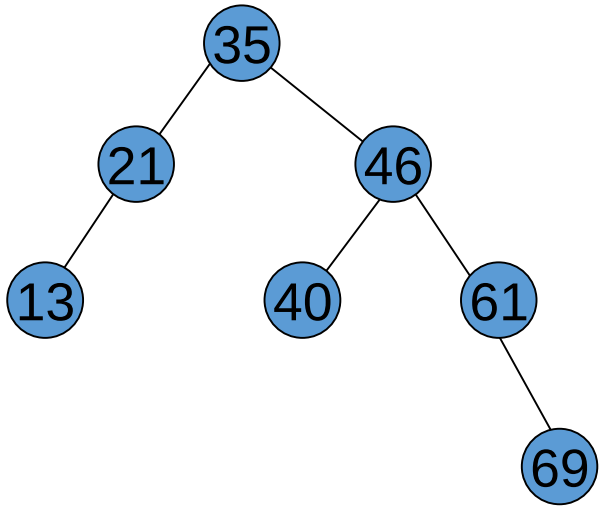
```
#include "stdio.h"
#include <math.h>
struct  BTNode
{
  BTNode *left;
  BTNode *right;
  int value;
};
struct BTNode *new_node(int x, struct BTNode *y, struct BTNode    *z)
{
  struct BTNode *w = new BTNode();
  w->value = x;
  w->left = y;
  w->right = z;
  return w;
}
int _tmain()
{
  int ch;
  BTNode *root;
  root = new_node( 100, NULL, NULL );
  printf( "Enter キーを1,2回押してください. プログラムを終了します¥n");
  ch = getchar();
  ch = getchar();
  return 0;
}
```

メンバ value, left, right に値をセット

例題２，３，４で使用

例題２．二分探索木の生成

- データの配置のしかた
  - 左側のすべての子は親より小さく
  - 右側のすべての子は親より大きく

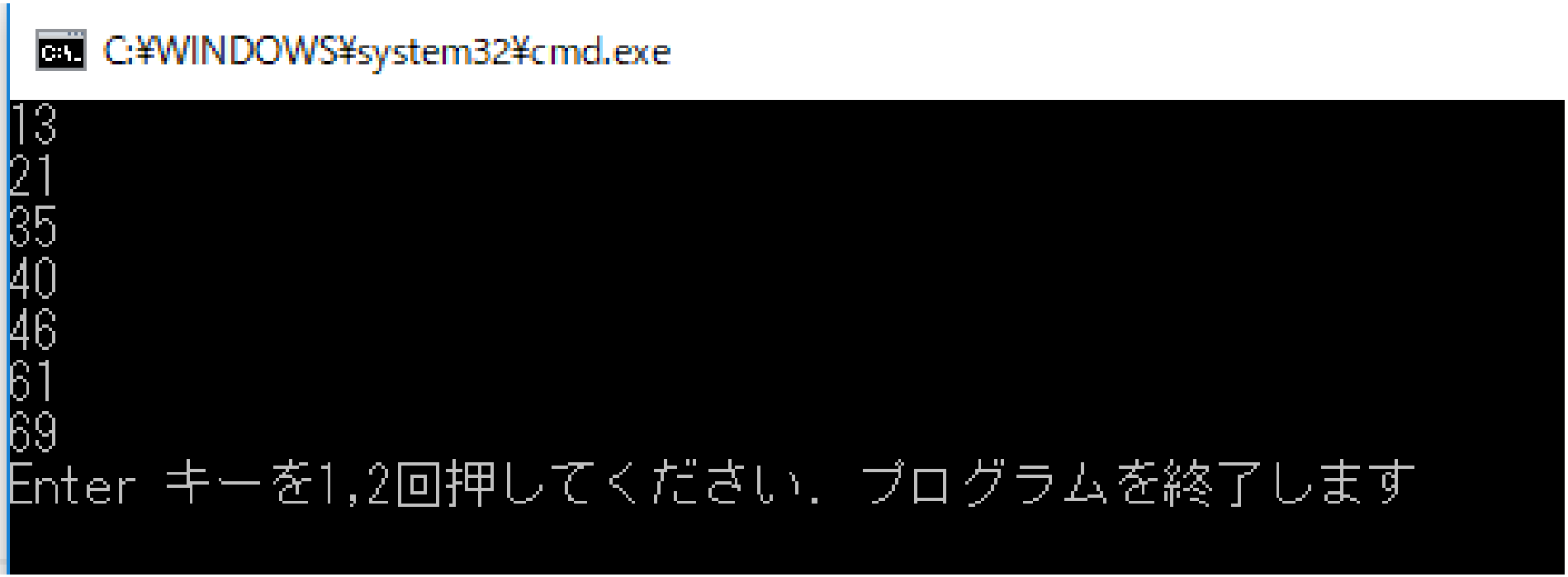


new_node を７回呼び出して，節点を7個作る

```
#include "stdio.h"
#include <math.h>
struct  BTNode
{
  BTNode *left;
  BTNode *right;
  int value;
};
void print_data( struct BTNode *root )
{
  if ( root->left != NULL ) {
    print_data( root->left );
  }
  printf( "%d¥n", root->value );
  if ( root->right != NULL ) {
    print_data( root->right );
  }
}
struct BTNode *new_node(int x, struct BTNode *y, struct BTNode *z)
{
  struct BTNode *w = new BTNode();
  w->value  = x;
  w->left  = y;
  w->right = z;
  return w;
}
int _tmain()
{
  int ch;
  BTNode *root;
  root = new_node( 35,
                new_node( 21,
                        new_node(13, NULL, NULL),
                        NULL),
                new_node( 46,
                        new_node(40, NULL, NULL),
                        new_node(61,
                                NULL,
                                new_node(69, NULL, NULL))));
  print_data( root );
  printf( "Enter キーを1,2回押してください. プログラムを終了します¥n");
  ch = getchar();
  ch = getchar();
  return 0;
}
```

in-order で要素を表示
（in-order については後述）

２分木の生成

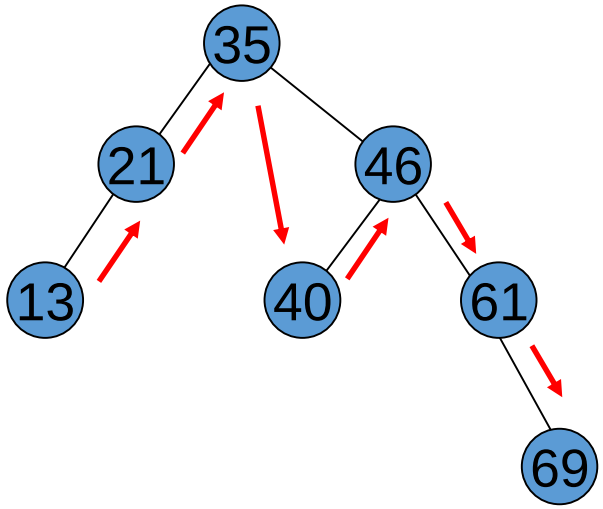

C:¥WINDOWS¥system32¥cmd.exe
13
21
35
40
46
61
69
Enter キーを1,2回押してください．プログラムを終了します

実行結果の例

- in-order での表示

```
void print_data( struct BTNode *root )
{
  if ( root->left != NULL ) {
    print_data( root->left );
  }
  printf( "%d¥n", root->value );
  if ( root->right != NULL ) {
    print_data( root->right );
  }
}
```

最初に左部分木
次に自分（節点）
最後に右部分木

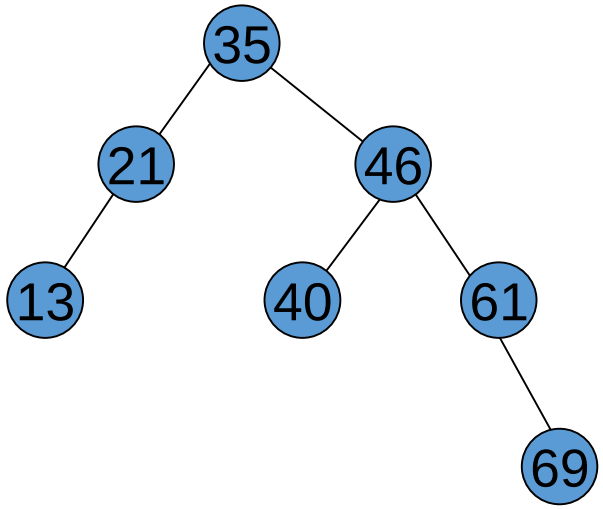


例題３．二分探索木における探索

- 指定された要素が存在するかを探す

４０　→　有り

４１　→　無し



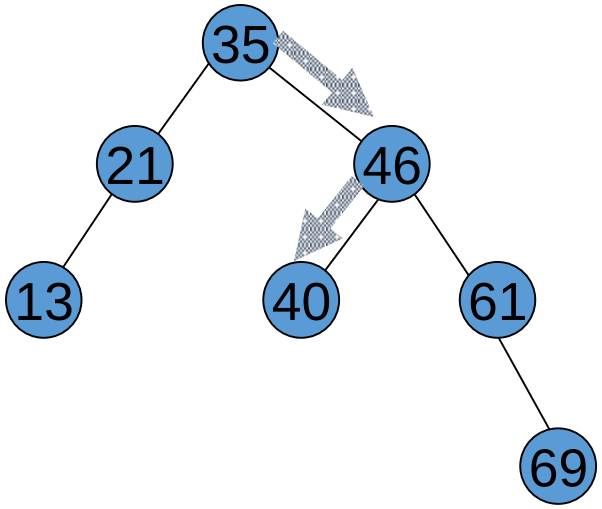
二分探索木における探索

- 根(root)から始める。
- 探索キーの値と，各節点のデータを比較し目標となるデータを探す
  - 探索キーよりも節点のデータが小さいときは，右側の子に進む
  - 探索キーよりも節点のデータが大きいときは，左側の子に進む

探索の例

（例） 節点40を探す場合

- 根の値(35)と，探索キー(40)を比較
- 探索キーの方が大きいので，右側の子節点へ移る
- 次に移った節点の値(46)と探索キー(40)を比較し
- 探索キーの方が小さいので，左の子節点へ移る
- 次に移った節点(40)が，目標の節点である



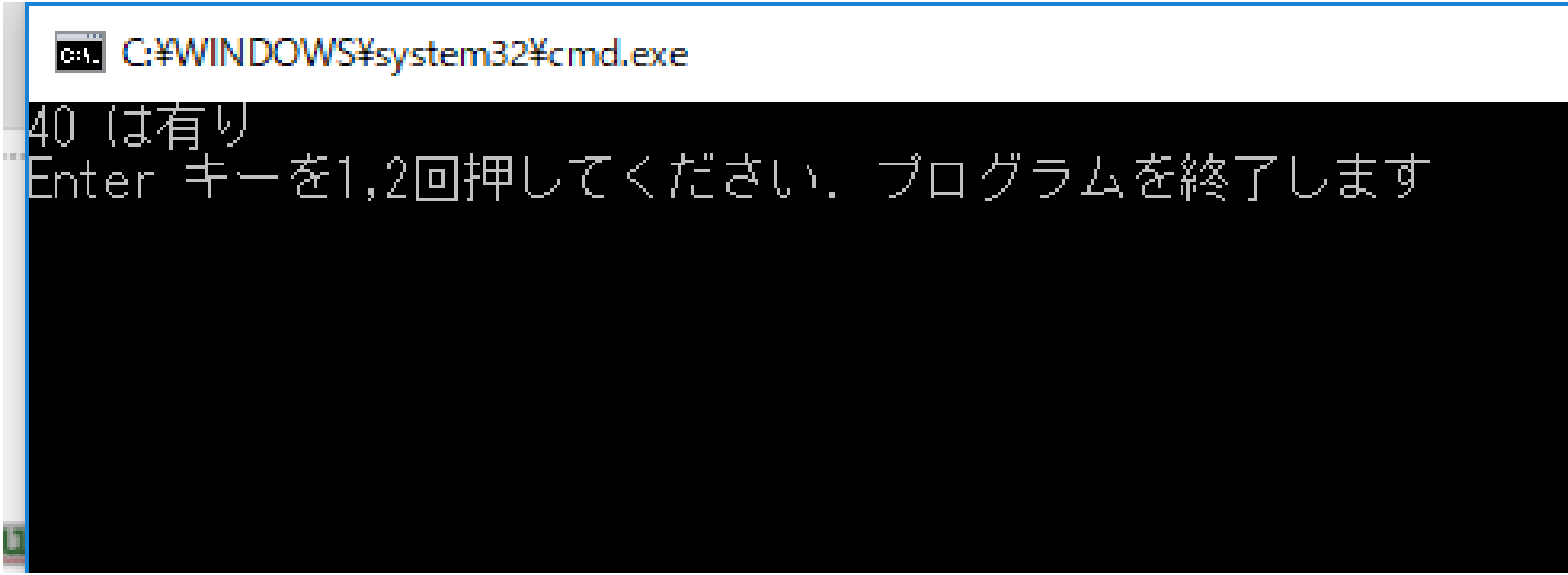
```
#include "stdio.h"
#include <math.h>
struct  BTNode
{
  BTNode *left;
  BTNode *right;
  int value;
};
struct BTNode *find_node(int x, struct BTNode *root)
{
  BTNode *node;
  node = root;
  while( ( node != NULL ) && ( x != node->value ) ) {
    if ( x < node-> value ) {
      node = node->left;
    }
    else {
      node = node->right;
    }
  }
  return node;
}
struct BTNode *new_node(int x, struct BTNode *y, struct BTNode *z)
{
  struct BTNode *w = new BTNode();
  w->value  = x;
  w->left  = y;
  w->right = z;
  return w;
}
int _tmain()
{
  int ch;
  BTNode *root;
  BTNode *y;
  root = new_node( 35,
                new_node( 21,
                        new_node(13, NULL, NULL),
                        NULL),
                new_node( 46,
                        new_node(40, NULL, NULL),
                        new_node(61,
                                NULL,
                                new_node(69, NULL, NULL))));
  y = find_node( 40, root );
  if ( y == NULL ) {
    printf( "40 は無し¥n" );
  }
  else {
    printf( "40 は有り¥n" );
  }
  printf( "Enter キーを1,2回押してください. プログラムを終了します¥n");
  ch = getchar();
  ch = getchar();
  return 0;
}
```

探索（見つからなければ NULL を返す）

2 分木の生成

探索を行う関数

```
struct BTNode *find_node(int x, struct BTNode *root)
{
  BTNode *node;
  node = root;
  while( ( node != NULL ) && ( x != node->value ) ) {
    if ( x < node-> value ) {
      node = node->left;
    }
    else {
      node = node->right;
    }
  }
  return node;
}
```


C:¥WINDOWS¥system32¥cmd.exe
40 は有り
Enter キーを1,2回押してください．プログラムを終了します

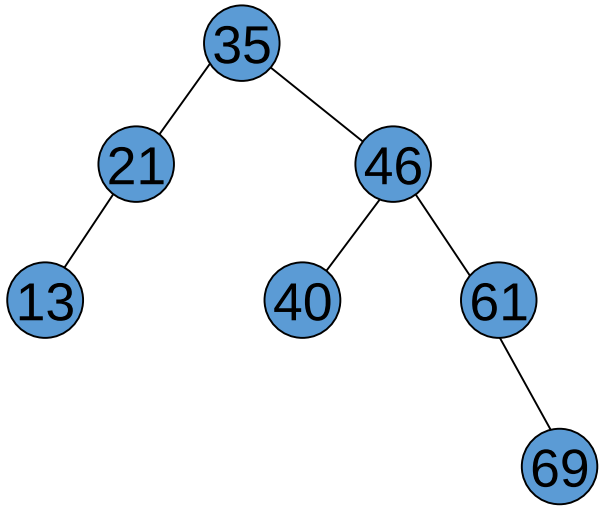
実行結果の例

例題４．二分探索木への挿入

- 要素の挿入を行う関数を作る

  35 46 21 13 61 40 69 を挿入すると，

  例題１と同じ二分探索木ができる



二分探索木への挿入

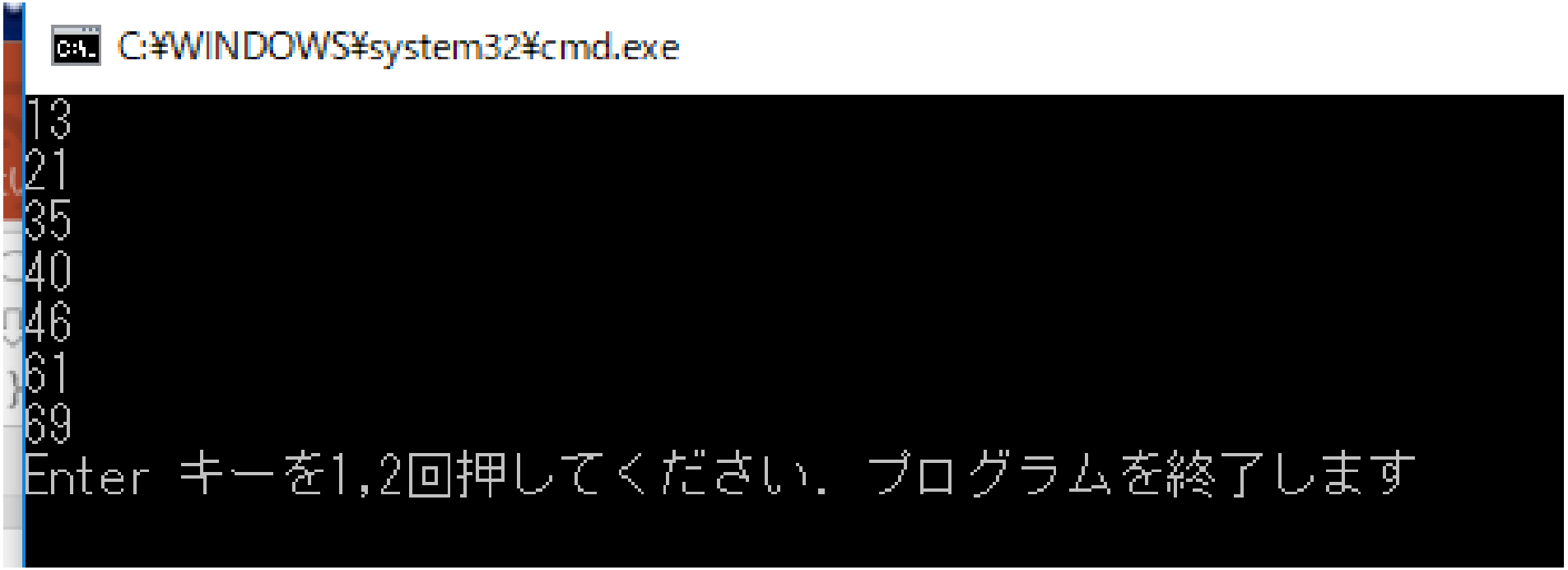
- 二分探索木に新たなデータを挿入する
  - 挿入すべき位置を探す
  - （find_nodeと同じ要領）
  - 新たな節点を生成
  - 挿入位置として見つかった節点において、新たな節点をポイントするようにポインタを書き換える

```
#include "stdio.h"
#include <math.h>
struct  BTNode
{
  BTNode *left;
  BTNode *right;
  int value;
};
void print_data( struct BTNode *root )
{
  if ( root->left != NULL ) {
    print_data( root->left );
  }
  printf( "%d¥n", root->value );
  if ( root->right != NULL ) {
    print_data( root->right );
  }
}
struct BTNode *new_node(int x, struct BTNode *y, struct BTNode *z)
{
  struct BTNode *w = new BTNode();
  w->value  = x;
  w->left  = y;
  w->right = z;
  return w;
}
struct BTNode *insert_node(struct BTNode *node, int x)
{
  if ( node == NULL ) {
    return new_node(x, NULL, NULL);
  }
  else if ( x < node->value ) {
    node->left = insert_node(node->left, x);
    return node;
  }
  else if ( x > node->value ) {
    node->right = insert_node(node->right, x);
    return node;
  }
  else {
    return node;
  }
}
int _tmain()
{
  int ch;
  BTNode *root;
  root = new_node( 35, NULL, NULL );
  insert_node( root, 46 );
  insert_node( root, 21 );
  insert_node( root, 13 );
  insert_node( root, 61 );
  insert_node( root, 40 );
  insert_node( root, 69 );
  print_data( root );
  printf( "Enter キーを1,2回押してください. プログラムを終了します¥n");
  ch = getchar();
  ch = getchar();
  return 0;
}
```

最初は，節点を１個含む二分探索木を作り，
その後，残りの要素を挿入する

挿入を行う関数

```
struct BTNode *insert_node(struct BTNode *node, int x)
{
  if ( node == NULL ) {
    return new_node(x, NULL, NULL);
  }
  else if ( x < node->value ) {
    node->left = insert_node(node->left, x);
    return node;
  }
  else if ( x > node->value ) {
    node->right = insert_node(node->right, x);
    return node;
  }
  else {
    return node;
  }
}
```

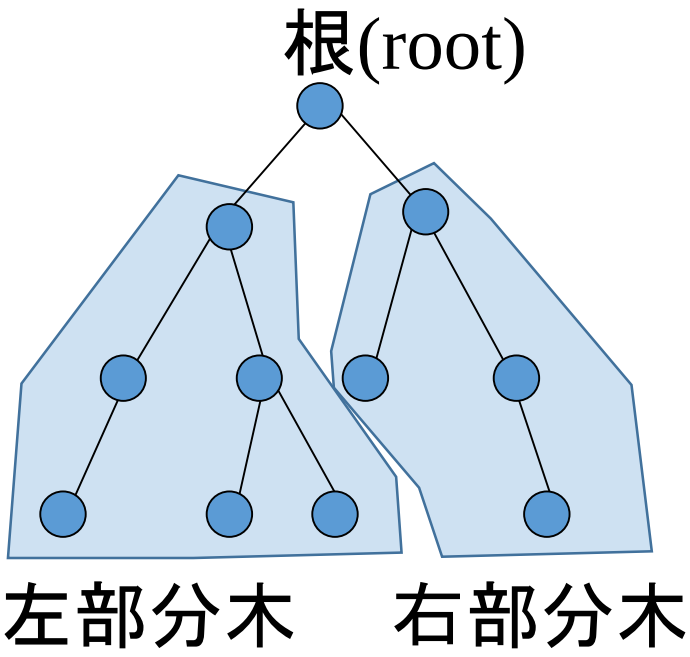

C:¥WINDOWS¥system32¥cmd.exe
13
21
35
40
46
61
69
Enter キーを1,2回押してください．プログラムを終了します

実行結果の例

二分木とは

レコードを次の３つで構成

- 要素を格納する**セル**
- **左部分木**を指す**ポインタ**を格納する**セル**
- **右部分木**を指す**ポインタ**を格納する**セル**



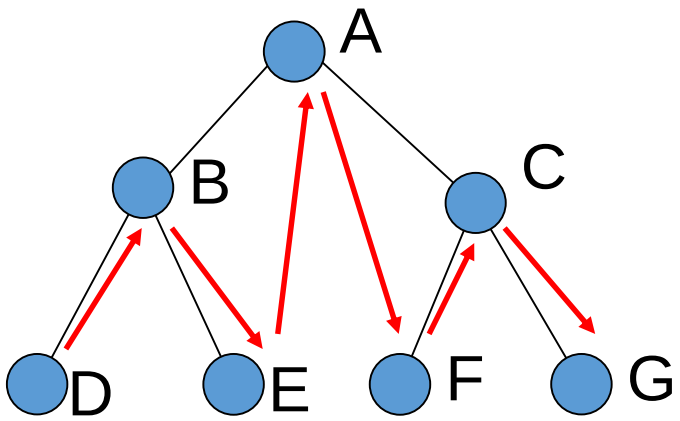
二分木の走査法

二分木の走査には次の種類がある

- 先行順 (pre-order)
- 中間順 (in-order)
- 後行順 (post-order)

先行順 (pre-order)

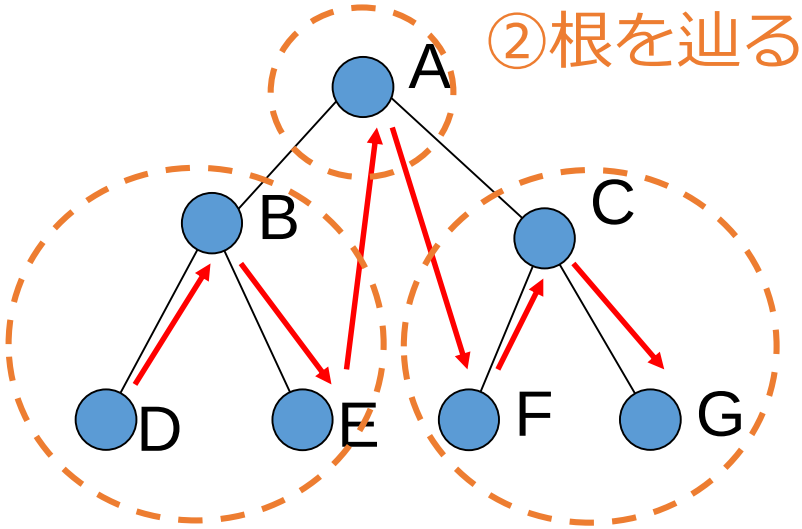
- 根ノード，左部分木，右部分木の順



D, B, E, A, F, C, G
の順に処理を行う

中間順 (in-order)

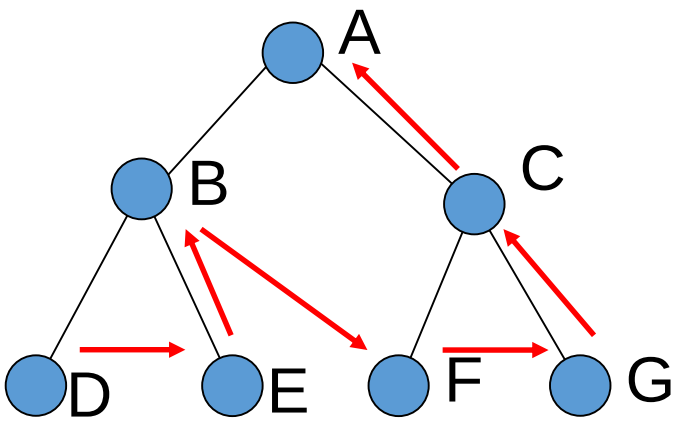
- 左部分木，根ノード，右部分木の順



D, B, E, A, F, C, G
の順に処理を行う

後行順 (post-order)

- 左部分木，右部分木，根ノードの順



D, E, B, F, G, C, A
の順に処理を行う